Automated Defence Intelligence systems are vulnerable to adversarial attacks

Intuition on the subject may be catastrophic

Available as a podcast

Defense intelligence is increasingly becoming a domain where computers play a pivotal role. This shift offers numerous advantages, as computers can swiftly process vast amounts of intelligence that would require human analysts days to analyze. Consequently, faster insights often translate into more actionable insights. However, this transformation also introduces risks, as intelligence is constantly susceptible to counterintelligence measures. Adversarial attacks have been demonstrated against various models, including image classification models and large language models (LLMs)[1]. These perturbations are often imperceptible to humans and can have severe consequences when applied to critical questions such as the likelihood of invasion.

Despite the potential significance of this field, public research in this area appears to be limited. While it is possible that this scarcity stems from the secretive nature of defense research, I did identify instances of adversarial attack literature in other domains of defense technology[2]. Notably, I could not locate a single reference to any paper discussing adversarial attacks specifically within the intelligence space.

Theoretical Framework

Let us imagine a simple system. The system is able to observe intelligence reports, evaluate evidence, and make a decision as to what course of action to take. That decision would likely be based on the strength of the evidence, the severity of the threat, and maybe external knowledge of the situation. The concept of a threat severity, however, is inseparable from the strength of the evidence. The consequences of an eruption of Yellowstone might be armageddon but, since the probability of this event is low, most of us don't

[1]Adversarial attacks in LLMS Adversarial attacks in Image Classifiers

[2] military adversarial AI

worry about it. External knowledge of the situation follows suit. If knowledge of the situation changes the probability of the event occurring, it necessarily changes the threat level. So perhaps the system only needs to evaluate the 'threat level' *provided that the threat level itself contains the other relevant information.*

If we were to build such a system, how would we do it? Such a system would need to be able to handle extremely complex relationships. It would need to be able to understand how different pieces of evidence interact with each other, how different pieces of evidence in different contexts paint vastly different pictures. Evaluating highly complex relationships is a specialty of neural networks. Training such a model would require a training set. A set of intelligence reports, with the labeled 'threat level' in which to learn classification. In this case we would consult domain experts, ask them to read these reports and suggest a threat level given the evidence. We would also have the benefit of hindsight, and could adapt threat levels based on actual outcomes. This does however risk overfitting. Sometimes the evidence is simply incomplete, and the model would likely look to noise in order to explain away the differences. There are also times however, when this may be useful. Perhaps a neural network may train itself not to trust evidence provided by a taxi driver who claimed to have overheard military personnel discussing WMDs (yes that was a genuine contributing factor to the decision to invade Iraq)[3].

Counter Intelligence in the age of AI

Counter Intelligence is almost as old as intelligence itself. In this context, I will refer to counter intelligence as the strategic use of deception and misinformation to manipulate an adversary's intelligence efforts, protecting sensitive operations and misleading enemy decision-making to gain a tactical or strategic advantage. Counter intelligence is even discussed in Sun Tzu's famous text 'The Art of War' in which he describes having 'double agents' that pretend to work for the enemy and actually feed them misinformation[4].

[3] 45-minute WMD claim 'may have come from an Iraqi taxi driver'

[4] Sun Tau in contemporary Chinese Strategy

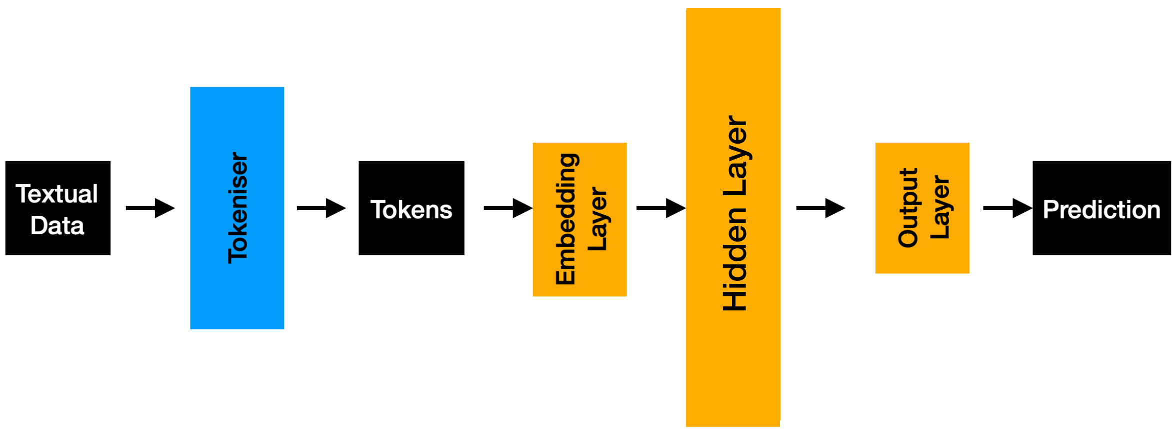
Let’s return to the example of the model that identifies a threat level. Through the lens of counter intelligence, there is a very specific field of AI research that would be of great use. Adversarial attacks on neural networks are deliberate manipulations of input data designed to deceive the model into making incorrect predictions or classifications. These attacks exploit vulnerabilities in the model by adding subtle, often imperceptible, perturbations to the input, causing the network to produce erroneous or unexpected outputs. In the context of espionage this might be a slightly different colored few pixels in an image of a building that leads a model to misclassify a building as an apartment complex when in fact it’s a military base. Or slight perturbations to the words describing the credibility of the source, ‘trustworthy’ becomes ‘reliable’. These attacks work by nudging the inputs in the opposite direction of the loss function gradient. Instead of minimising loss we instead maximise it. For a more in depth explanation see this paper.

Practical Implementation

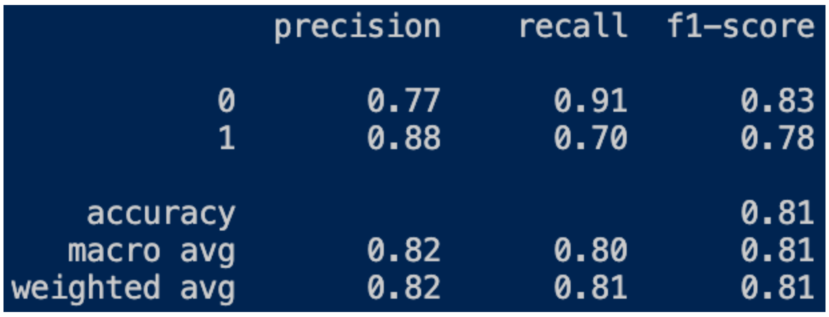
Intelligence reports have one thing that makes them incredibly difficult to train models on, they’re classified. Outside of some very rare cases that reports are declassified to explain decisions to the public, most remain under lock and key. Fortunately, there is a large corpus of declassified CIA intelligence reports available here. Unfortunately, domain experts in this field are too busy trying to predict Russia’s next move in Ukraine to go through and analyse old reports with different threat levels. For this reason, I undertook the role of threat level classification myself. The resulting dataset is available on Kaggle.

There are some serious limitations with this dataset. First is the size, It’s only ~100 reports. This was a practical limitation of the number of reports I was able to read and classify. The second limitation is, well, me. I’m not a domain expert, the extent of my foreign policy knowledge is the ‘world’ sections of the Financial Times and the New York Times. There is undoubtedly important information embedded in this report that I did not pick up on. For this reason I limited myself to ‘High’ and ‘Medium’ threat classifications, which in practice became ‘High’ and ‘not high’. The final limitation is the time period. The reports almost all hail from the cold war era, a fact that is extremely obvious in some of the word clouds I created (see Figures on Kaggle). This means that any model is likely to overfit to the time period, picking up on words like 'Soviet' and 'Communist' more than would be useful in the modern era.

Given these limitations, we should focus on the 'big picture' rather than the very obvious limitations of the model and it's excessive vulnerability to adversarial attacks. With that in mind, I will introduce you to Espion-AI (catchy name is very important in AI). This is a neural network trained on the data and labels from the Kaggle database I discussed earlier. The model takes tokenised text using the "basic English" tokeniser in PyTorch, and passes it first to an embedding layer. This is subsequently passed to a hidden layer before finally being passed to an output layer for a binary classification, High Threat (1) or Low Threat (0). A Diagram is included below:



The tokeniser was limited in length to the 50th percentile of the lengths produced by all the input intelligence reports. This both standardised input lengths and also reduced overfitting as especially long reports were not immediately recognisable. Given that much of the important information was probably in the 'conclusions' section, there is probably a better way of doing this. An accuracy of ~80% was achieved. While this is far from what we would want to see before deploying, it does demonstrate the immense power of these classifiers to learn complex relationships even when under immense constraints (computational and dataset length). Hyper-parameters seemed to have very little effect on model performance, embedding dimensions, number of epochs and hidden layer size only seemed to cause overfitting. Here is an example classification report I obtained with just 3 embedding layers.

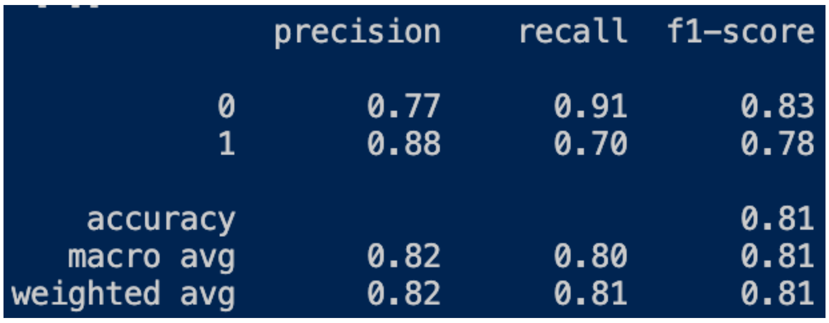


```
              precision    recall  f1-score

           0       0.77      0.91      0.83
           1       0.88      0.70      0.78

    accuracy                           0.81
   macro avg       0.82      0.80      0.81
weighted avg       0.82      0.81      0.81
```

Versus a report from a model with 120 dimensions:



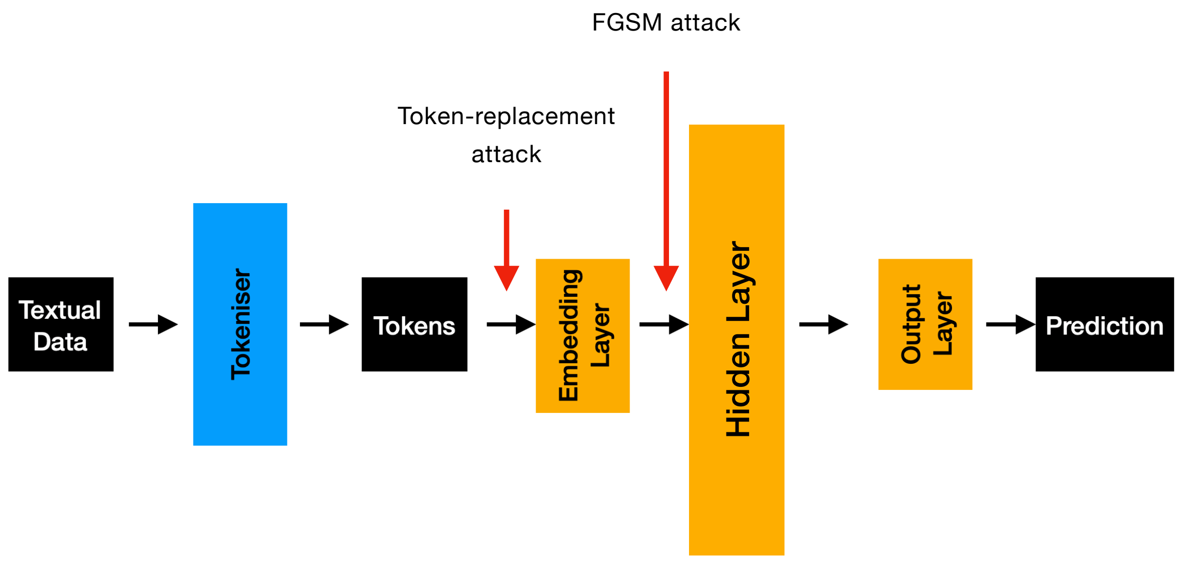
```
              precision    recall  f1-score

           0       0.77      0.91      0.83
           1       0.88      0.70      0.78

    accuracy                           0.81
   macro avg       0.82      0.80      0.81
weighted avg       0.82      0.81      0.81
```

It should also be noted that the model was quite poor at identifying High threat levels when they existed, not something we would want in this type of system, but again I emphasize the importance of not getting lost in the specifics of *this* model, and think instead about the concept of *a* model that performs this task. I highly encourage you to look at the source code to see specifics of how I trained the model, and even play around with it yourself.

For the adversarial attack, I used a method called fast gradient sign method or FGSM. This is a very popular method for adversarial attacks, which is documented extremely well here. This was particularly challenging due to the embedding layer. FGSM works by applying gradients directly to the input. In image models, this is straightforward because the input (e.g., pixel values) is continuous. In embedding layers, however, the input is discrete (indices), and FGSM can't directly modify these indices. Two workarounds presented themselves. First was to replace the tokens with semantically similar tokens based on gradients. The second was to apply the FGSM only to the model after the embedding layer. The two options are demonstrated below:

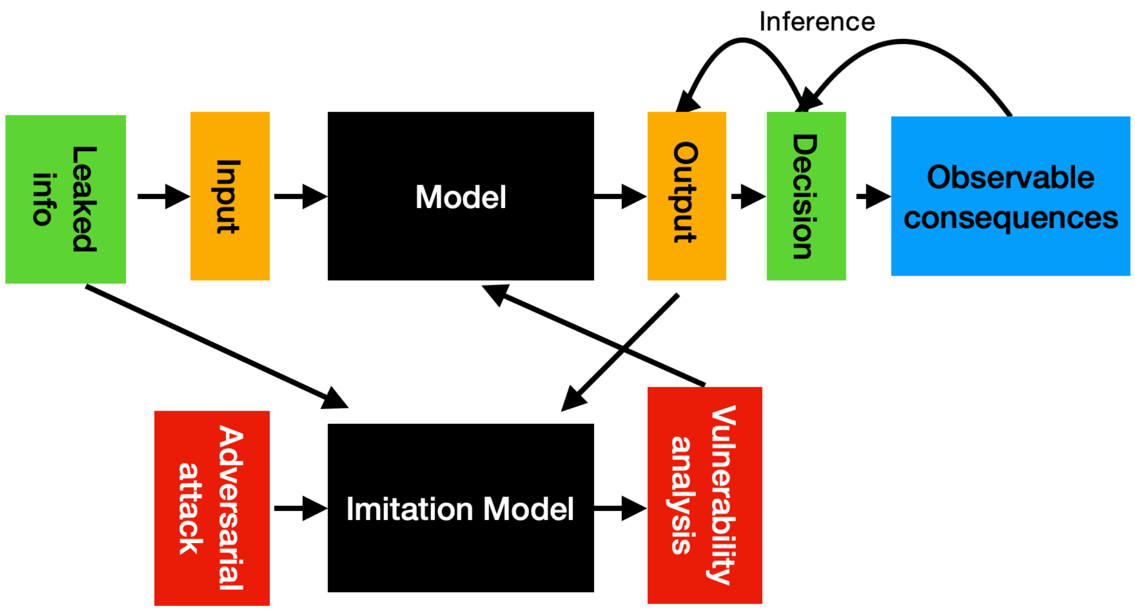


The first is certainly preferable. It has the advantage of being able to observe the differences, we could observe the differences in the input text and it would be targeting the entire model. However, this unfortunately stretched beyond my technical capabilities at this stage, I hope to implement this in future iterations of this project. The implemented code however, uses the second method. This sacrifices interpretability but nonetheless proves the vulnerability of these systems to attacks. The code is available in the same repository, and you can use the adversarial.py script to see the model being tricked, I was able to see incorrect classifications with epsilon values as low as 0.03.

The first instinct is incorrect

The first instinct to the following results is to lock the model up. If our models can be attacked adversarially we should make sure to keep them as secretive as possible, make sure only the highest security clearances have access to them. Adversarial attacks work by nefariously probing the model to find its weak spots and exploit them, so we can avoid this by limiting probing. Some of this is probably good practice, these models shouldn't be open source (the data on the other hand probably should be). I think the risk here is that explainability is rejected out of premise. For example one may argue against mechanistic interpretability of these models on the premise that it is better no one knows what makes them tick, thus avoiding leaks that could be harmful. I think this would be a great mistake.

We cannot avoid adversarial attacks on these models. Imagine a perfectly secret model, only one person feeds it inputs, and a different person reads its output. It's a totally opaque black box and no one knows anything about how it works. The people discussed are completely unbreakable by interrogation or any other workaround you can come up with. Now imagine I'm a nefarious actor. Returning to our counter-intelligence mindset, intelligence could be intentionally leaked to the operators of this model, real or otherwise we just have to get it to the model. This model will then presumably be used to make decisions in the real world that can be observed. Place troops here, send tanks to this country, visit this head of state to strengthen ties. From these decisions can be inferred an output from the model. We now have a set of inputs and outputs. From this we can train our own imitation-model. Then we can run adversarial attacks on this model that will work on the original model. I have included a diagram below to try and describe what I mean.



The solution

So we're all doomed. Obviously not, but the solution is more complex than the knee jerk reaction. What we should actually do is interrogate these models to the fullest extent of the explainable methods techniques. That way model weaknesses and vulnerabilities can be understood and either corrected for or at least taken into account when decisions are being taken based on model outputs. It should also be assumed that adversarial attacks will be taking place. Such attacks should take place in training as well, adversarial training is a well established technique to improve model robustness but is extremely key in this case. Ensembling could also be employed to reduce the dependence on individual models that could be victims of attacks. There are a wealth of techniques available to defend against such attacks, but being aware of these risks is extremely important.